AD-A067 799 NAVAL SURFACE WEAPONS CENTER DAHLGREN LAB VA F/G 9/2
AN AUTOMATED SOFTWARE MAINTENANCE TOOL FOR LARGE-SCALE OPERATIN--ETC(U)
DEC 78 A L ZIRKLE

UNCLASSIFIED NSWC/DL-TR-3936 NL

END
DATE FILMED
6-79
DDC

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: NSWC/DL-TR-3936	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE *(and Subtitle)*: AN AUTOMATED SOFTWARE MAINTENANCE TOOL FOR LARGE-SCALE OPERATING SYSTEMS	5. TYPE OF REPORT & PERIOD COVERED: Final rept.
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): Alan L. Zirkle	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Naval Surface Weapons Center, Dahlgren Laboratory, K74, Dahlgren, VA 22448	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: Computer Program Support
11. CONTROLLING OFFICE NAME AND ADDRESS: Navy Industrial Fund	12. REPORT DATE: December 1978
	13. NUMBER OF PAGES: 30
14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)*: 32p.	15. SECURITY CLASS. *(of this report)*: UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

DDC RECEIVED APR 26 1979 A

18. SUPPLEMENTARY NOTES

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

Operating Systems	Software Development Methodology
Programming	Software Maintenance
Program Libraries	Software Tools
Programmer Efficiency	Systems Programming

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

A method for automating many of the tasks involved in maintaining a large computer operating system (SCOPE 3.4 on the CDC 6700) is described. The method is embodied in several procedures, each of which aids in one phase of operating system maintenance. These procedures are written in a manner that promotes ease of modification or enhancement. This automated maintenance tool can also be used in other programming applications where a large amount of effort must be expended in noncreative "housekeeping" tasks.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

391 598

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

79 04 25 009

NSWC/DL TR-3936
December 1978

AN AUTOMATED SOFTWARE MAINTENANCE TOOL FOR LARGE-SCALE OPERATING SYSTEMS

by

Alan L. Zirkle
Strategic Systems Department

NAVAL SURFACE WEAPONS CENTER
DAHLGREN LABORATORY

Dahlgren, Virginia 22448

FOREWORD

This report describes a method for automating many of the tasks involved in the complex process of maintaining a large computer's operating system. The method was designed and implemented for use with the SCOPE 3.4 operating system released by the Control Data Corporation for their 6000 series of computers.

This automated maintenance tool was developed in the Programming Systems Branch (K74) of the Computer Programming Division. This report was reviewed by Mr. Hermon Thombs, Head of the Programming Systems Branch.

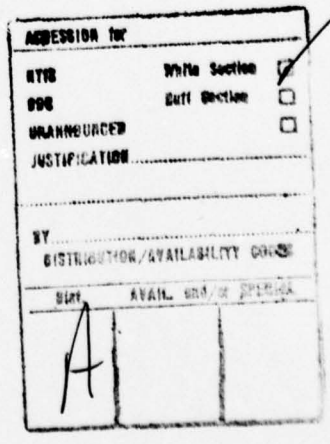
Released by:

Ralph A. Niemann

RALPH A. NIEMANN, Head
Strategic Systems Department

ACKNOWLEDGEMENTS

Development of this maintenance tool would not have been possible without the inspiration of Burnie Meyer, of Control Data, or without the help of the entire K74/CDC Systems Programming Group, who debugged the procedures and suggested many valuable improvements.



CONTENTS

LIST OF ILLUSTRATIONS

INTRODUCTION

The Naval Surface Weapons Center (NSWC), Dahlgren Laboratory, operates two large general-purpose computers built by the Control Data Corporation: a CYBER 70 model 74 system, and a 6700 multiprocessor system. Each of these two computers operates under the control of the SCOPE 3.4 operating system. SCOPE is a group of programs and subprograms that controls the input, compilation, loading, execution, and output of all programs submitted to the computers. A knowledge of SCOPE 3.4, and of SCOPE systems programming methods, will be helpful in understanding the material presented in this report.

Although the SCOPE 3.4 operating system is a Control Data product, its implementation at NSWC contains numerous locally installed modifications, known generically as "local mods." A local mod is introduced into a program to correct known errors in the program, to enhance its performance, or to eliminate inefficiencies.

Generation of a local mod is an exacting task, requiring significant amounts of expensive resources such as dedicated computer time and systems programmer manpower. This report describes a method for automating many of the tasks involved in creating and maintaining local mods, which results in increased productivity. This automation tool is also applicable to other Control Data operating systems (NOS and NOS/BE) and to programming tasks other than operating systems maintenance.

No automated method can substitute for the creativity of an experienced programmer; many necessary programming tasks, however, are essentially noncreative and repetitive in nature. An example of such a task is the installation of a local mod after the mod has been designed. The installation process (Figure 1) involves coding and executing an "installation deck" (described in the next section). Errors in an installation deck can cause substantial amounts of systems programmer manpower and computer time to be wasted; in extreme cases, incorrect implementation of a mod can result.

Fortunately, the installation process, including the coding of the installation deck, can be automated extensively. This is accomplished by using BEGIN/REVERT,* a high-level command language facility. BEGIN/REVERT allows conditional execution of control statements and provides for storing "subroutines" of control statements on files known as "procedure files."

The automated installation process, called the "COMPILE procedure," is actually a mixture of BEGIN/REVERT procedures and FORTRAN programs. FORTRAN is used because BEGIN/REVERT is limited in its ability to express complicated logical expressions. Procedure files and data files are created by FORTRAN programs and executed via the BEGIN/REVERT facility.

---

* Under NOS and NOS/BE, CYBER Command Language (CCL) is used. BEGIN/REVERT was developed by the University of Washington, Seattle, Washington.

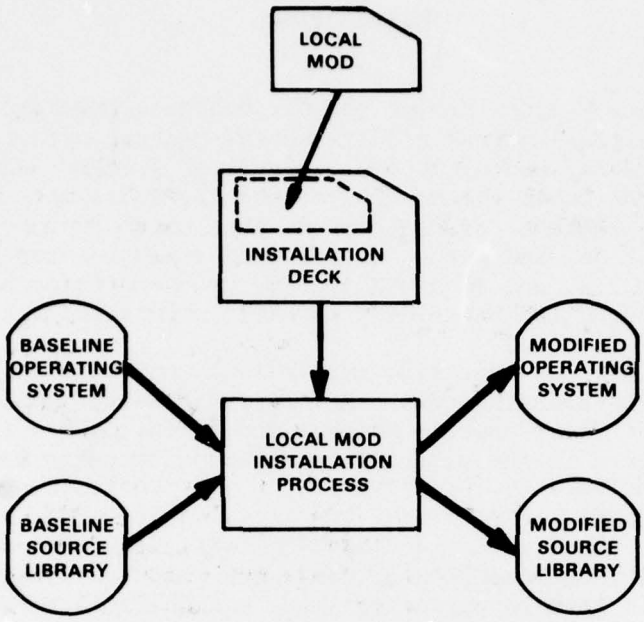


Figure 1. Data Flow in Installing a Local Mod

USING THE SOFTWARE MAINTENANCE PACKAGE--INSTALLING LOCAL MODIFICATIONS

The SCOPE 3.4 operating system consists of several hundred distinct program modules. Each program module has unique characteristics that may affect the implementation of changes to the program. Differences between programs include the following types:

Structural differences. Programs can be peripheral processor programs or central processor programs. They can be divided into multiple overlays.

Location differences. Programs can be on one of the program libraries provided by Control Data, or on a locally originated source library.

Translation differences. Programs may be coded in assembly language or FORTRAN (even SYMPL, COBOL, or PASCAL). One or more system text overlays may be required for assembly.

Binding Differences. Programs may be relocatable or absolute; link-editing with other modules may be required.

The characteristics of a program are usually constant over the lifetime of the program. The installation of a program, as expressed in the contents of the installation deck, is, of course, highly dependent on these characteristics. For example, note the differences (and similarities) in the installation decks of PFA (Figure 2) and RBTC (Figure 3).

```
. . .

PAUSE.      MOUNT EDITLB, PLSPAK.
MOUNTT,EDITLB.
ATTACH,LOCALPL,SN=EDITLB.
UPDATE,P=LOCALPL,C=LOCAL,Q,*=/.
MOUNTT,PLSPAK.
ATTACH,PL1B,SN=PLSPAK.
UPDATE,P=PL1B,I=LOCAL,Q,X.
REQUEST,BPFA,*PF,SN=EDITLB.
COMPASS,I,S=IPTEXT,S=PPTEXT,S=SCMTEXT,B=BPFA.
CATALOG,BPFA, . . .
ITEMIZE,BPFA,E,N.
. . .

7-8-9
/ADDFILE
/DECK LOCAL
/CALL SC4BCOM
/CALL PFA

   <THE TEXT OF THE LOCAL MOD GOES HERE>
. . .
```

Figure 2. Part of the Installation Deck for PFA

```
. . .

PAUSE.      MOUNT EDITLB.
MOUNTT,EDITLB.
ATTACH,LOCALPL,SN=EDITLB.
UPDATE,P=LOCALPL,C=LOCAL,Q,*=/.
ATTACH,PL1Z,SN=EDITLB.
UPDATE,P=PL1Z,I=LOCAL,Q.
REQUEST,BRBTC,*PF,SN=EDITLB.
FTN,I,R,PL=9999999,S=CPCTEXT.
ATTACH,SYSLIB.
ATTACH,KPSLIB.
LIBRARY,SYSLIB,KPSLIB.
LDSET,PRESET=ZERO.
LOAD,LGO.
NOGO,BRBTC.
CATALOG,BRBTC, . . .
ITEMIZE,BRBTC,E,N.
. . .

7-8-9
/ADDFILE
/DECK LOCAL
/CALL PL1ZCOM
/CALL RBTC

   <THE TEXT OF THE LOCAL MOD GOES HERE>
. . .
```

Figure 3. Part of the Installation Deck for RBTC

Before the COMPILE procedure was implemented, the systems programmer had to either (*a*) punch such an installation deck for each local mod, which was discarded after the mod was processed, or (*b*) save a deck for each program, using it every time the program was modified. Of course, option (*a*) duplicated a lot of effort, but (*b*) was just as bad since it led to filing problems and errors when deck format changes were necessary.

Using the COMPILE procedure, the above two example decks are replaced by

```
(1)  BEGIN,COMPILE,,PFA.
(2)  BEGIN,COMPILE,,RBTC.
```

The COMPILE procedure then generates streams of control statements equivalent to those in Figures 2 and 3. Notice that the systems programmer is relieved of the responsibility of specifying any of the characteristics of the program being modified. He is also relieved of the need to do a lot of keypunching!

Replacing control statements with BEGIN/REVERT procedures, however, is not an exciting breakthrough. The significant feature of the COMPILE procedure is its ability to configure itself to process programs that have highly divergent characteristics.

How does the COMPILE procedure know the characteristics of all the programs in the operating system? The procedure contains a data base that describes every significant characteristic of every program in SCOPE. The data base is easily maintained because it is a card image file. Each card contains the characteristics for one program in a shorthand notation. For example, the cards for the PFA and RBTC programs are

```
PFA     BP  O5PA
RBTC    ZF  TB  LSK
```

which are read

> "PFA is located on PL1B (B). It is a peripheral processor program (P). It has multiple overlays, the last of which is 5PA (O5PA)."
>
> "RBTC is located on PL1Z (Z). It is a FORTRAN program (F). It needs system text CPCTEXT for compliation (TB), and user libraries SYSLIB and KPSLIB for loading (LSK)."

Part of the data base is a glossary that explains the notations used in the data base. The user of the COMPILE procedure can override any information in the data base by specifying optional parameters on the BEGIN command (see Appendix A, NSWC User's Guide for the Software Maintenance Package).

A FORTRAN program (also named COMPILE) within the COMPILE procedure creates the control card images which are ultimately executed to perform the installation. The COMPILE program finds out what SCOPE program is being installed (from the first parameter on the BEGIN command) and searches the data base for the

program's entry. Any optional parameters on the BEGIN command are used to override configuration values from the data base entry (using the rules explained in the user's guide), and the resulting values are used to build a new, temporary procedure file named CTLCDS, which is subsequently executed via a nested BEGIN call.

The COMPILE program, which builds the card images, contains by necessity logical equations of some complexity. Figure 4, for example, shows the flow of a minor part of the COMPILE program--the decision whether or not to generate a MOUNT command to mount the EDITLB device set. Maintainability of the COMPILE program is possible in spite of its complexity because the program is modular and well-documented with in-line comments.

EXPERIENCE WITH THE MAINTENANCE PACKAGE

As of November 1978, the maintenance package has been in production use for eight months. An informal class was given to all users (the K74 and CDC Systems Programming Group) when the package was introduced. A user's guide (Appendix A) was written and is revised periodically to reflect changes in the package or to meet user requests for more information.

The package itself has been modified several times to add or change features, following user's suggestions. The package has been enthusiastically accepted by the users, and it appears to be contributing to a more productive and stable environment of system modifications.

Although the COMPILE procedure has been highlighted in this report, the maintenance package includes other procedures used by the Systems Programming Group. They are described in the user's guide (Appendix A).

Appendix B discusses the SCOPE system utility programs used in installing local mods.

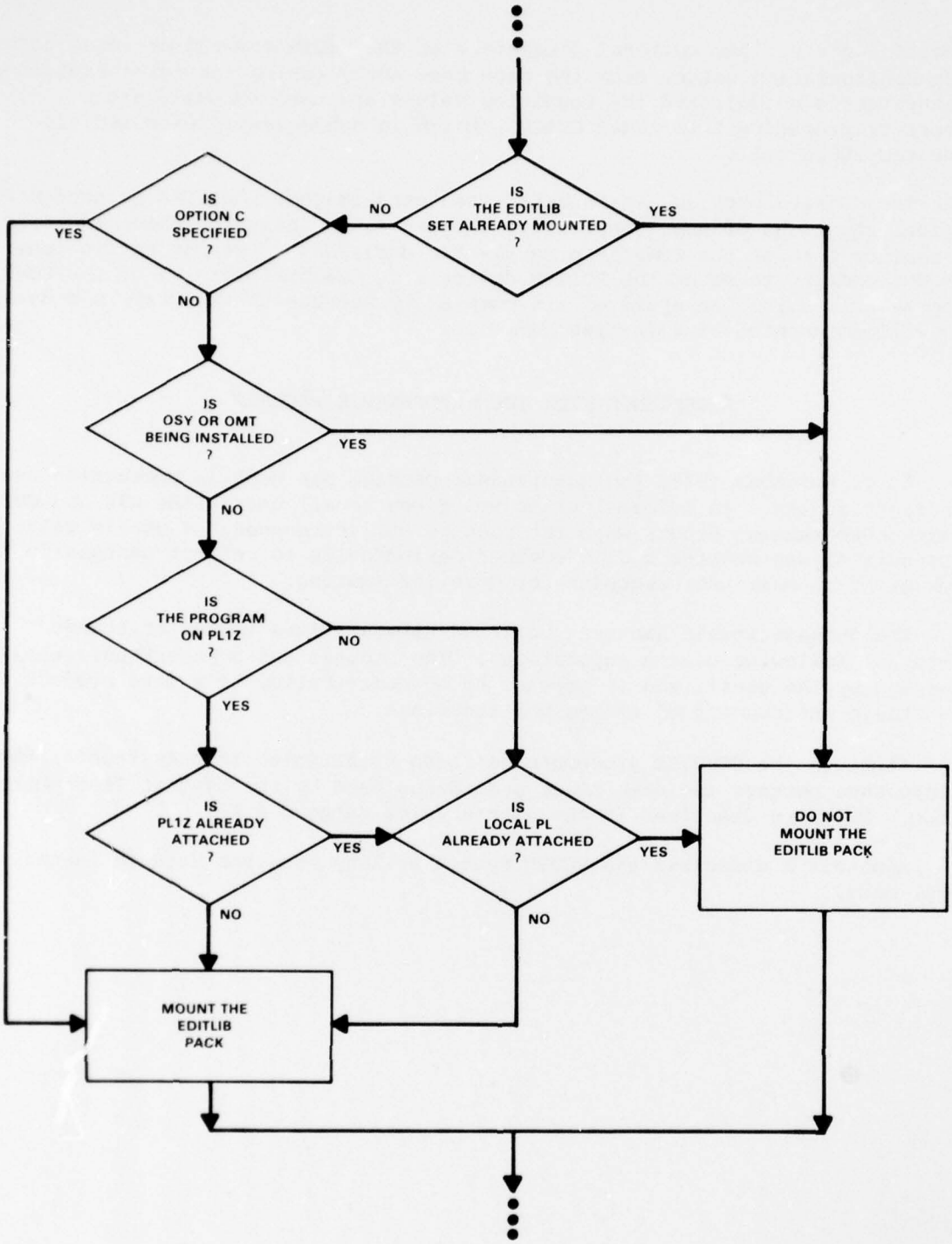


Figure 4. Logic Flow for the Decision to Mount the EDITLB Device Set

APPENDIX A

NSWC USER'S GUIDE

FOR THE SOFTWARE MAINTENANCE PACKAGE

APPENDIX A

NSWC USER'S GUIDE FOR THE SOFTWARE MAINTENANCE PACKAGE

This appendix is an example of the NSWC user's guide for the COMPILE procedure and other procedures in the maintenance package. This example has been edited to delete proprietary and sensitive information (i.e., passwords).

The user's guide is meant to be an informal document, which can be easily changed when features are added or when responses from the users indicate that a feature should be explained more clearly.

A knowledge of SCOPE 3.4 operating system installation methods and of NSWC's local modification process is necessary to fully understand the material in the user's guide.

COMPILE PROCEDURE

The COMPILE procedure generates the control cards, UPDATE directives, and EDITLIB directives necessary for modifying components of the SCOPE 3.4 operating system at NSWC/DL.

The Basic Calls

```
BEGIN,COMPILE,,prog,OPT=A.  (To get a listing only)
BEGIN,COMPILE,,prog,cy.
BEGIN,COMPILE,,prog,cy,ac.
```

where

- prog -- the name of the program to be compiled. This is a required parameter.
- cy -- the cycle number for cataloging the binary result. This number is required when OPT=C is in effect (default OPT is OPT=AC).
- ac -- programmer name to be used for the binary file's account code. This code is required when OPT=C is in effect, except for Systems Programming Group members, for whom it is optional (it is computed from the job card if not specified).

General Information

COMPILE gets the information needed to UPDATE, assemble, load, and EDITLIB the requested program from its data base, on the local file CDATA. Every program on PL1A, PL1B, PL12, PL1Z, and PL1T is represented on this data base. Any data base information can be overridden by specifying one or more of the optional parameters described below. If you have a local file CDATA, COMPILE will use your file as its data base. Similarly, if you have local files named

LOCALPL, PL1A, PL1B, PL12, PL1Z, or PL1T, COMPILE will UPDATE from your file instead of from the standard program library of the same name.

In addition, COMPILE will do all mounting and dismounting of the correct Device Sets. It will not mount a set until it is necessary, and will dismount a set as soon as it is no longer needed.

If Sense Switch 1 is on, COMPILE will not pause to ask the operator to mount the packs (this option is useful during System Time). If Sense Switch 2 is on, output from UPDATE will be printed in all cases (output is normally suppressed if UPDATE completes successfully).

The Optional Parameters

OPT = x -- where x can be one of the following:

A	--	assemble the program
AC	--	assemble and catalog the binary (default)
AE	--	assemble and EDITLIB the program
ACE	--	assemble, catalog, and EDITLIB

PL = x -- the program library on which "prog" resides; x must be PL1A, PL1B, PL1Z, PL1T, or PL12.

TXT = x -- specifies System Texts necessary for assembly. Texts specified by this parameter are exclusive or'ed with the data base value, which is then exclusive or'ed with the default. This final value is used in assembly. The defaults are as follows:

PL1A PP programs:	TXT = E
PL1A CP programs:	TXT = G
PL1B programs:	TXT = BCDEF
PL12 programs:	TXT = DEFH
PL1Z PP programs:	TXT = E
PL1Z CP programs:	TXT = M

The Texts are as follows:

A - CMRTEXT	H - STATEXT
B - CPCTEXT	I - 0 (use no Texts)
C - CPUTEXT	J - IOTEXT
D - IPTEXT	K - LDRTEXT
E - PPTEXT	L - PFMTEXT
F - SCHTEXT	M - SYSTEXT
G - SCPTEXT	

When I is in effect, it overrides all others; i.e., TXT=ABIKL is treated as TXT=I.

If the job has a local file named BTEXTS, then COMPILE will get the required texts from this file.

LIB = x -- libraries necessary for loading relocatable CP programs. These are exclusive or'ed with the data base values like the Texts are. The libraries are as follows:

S - SYSLIB	C - COBOL
K - KPSLIB	R - RUN2P3
U - USERLIB	I - SYSIO
D - DMS170	M - SYSMISC
O - SYSOVL	G - IGS274
F - FORTRAN	

The user must attach KPSLIB when it is needed; COMPILE will attach other requested user libraries, unless a local file of the same name already exists.

DECK = x -- used when the LOCALPL deck name (or the SCOPE PL deck name) is different from the program name. An example is 1Z2, which has a deck name of "1M1." This parameter usually only appears in the data base.

REL = R -- CP COMPASS programs are assumed to be absolute unless REL=R is specified, which causes a LOAD/NOGO to be performed. Don't use REL=R if the program must remain relocatable (e.g., QUEDUMP, SYSEQ, CPC).

SP = x -- an indicator that "prog" needs special processing in COMPILE because of nonstandard characteristics in its method of installation. This parameter usually only appears in the data base. An example of such a special program is 1Z2, which needs a "*DEFINE CT71" card in its UPDATE directives. To disable any SP value in the data base, SP=NONE can be specified on the BEGIN card for COMPILE.

VER = x -- program version indicator. See the section below on "Special Programs" for use of this parameter.

PS = x -- option for presetting core for CP programs that undergo a LOAD/NOGO. Default is PS=ZERO. If PS=NONE is specified, no presetting is done.

If the parameter begins with an asterisk (*), a PRESETA is performed instead of a PRESET. See the Loader Manual for details. Examples:

PS = 100B PS = *INDEF

OV1 = x -- name of the first overlay in "prog." Default is the value of "prog."

OV2 = x -- name of the last overlay in "prog." Default is the value of "OV1."

FL = n -- field length for EDITLIBing CP programs. If not specified, the current value will be unchanged.

The UPDATE Process

COMPILE does two UPDATEs. The first is against the LOCALPL with the file UPDIN as input, and LOCAL as the compile file. The second is against the SCOPE PL, with LOCAL as input.

COMPILE builds the file UPDIN. Its structure is as follows:

```
/ACCFILE
/DECK $COMDECKS
/CALL ____COM

<ENTIRE CONTENTS OF FILE 'DECKS'>

/ACCFILE
/DECK $$<DECK>
/IF DECK,<DECK>,1                    *
/CALL <DECK>                         *
/IF -DECK,<DECK>,1                   *
*COMPILE <DECK>                      *

<ENTIRE CONTENTS OF FILE 'IDENTS'>

<CONTENTS OF NEXT RECORD OF INPUT FILE>
```

where "<DECK>" is the value of the DECK parameter (which defaults to "prog").

The user can use the files DECKS and IDENTS to introduce UPDATE directives. If this is done, and if the HEADER procedure is called, care must be taken to never rewind these files, since HEADER also puts directives on these files. If no cards are to be read from the INPUT file, an empty record must be included (or, under INTERCOM, the file should be disconnected). The UPDATE listings are normally not printed unless an UPDATE error occurs.

Verify Mode

In Verify Mode, the control cards created by COMPILE (on the file CTLCDS) are listed, not executed. The UPDATE directives and EDITLIB directives on the file UPDIN are also listed. Verify Mode is automatically entered when COMPILE is executed from INTERCOM; it is entered from batch whenever a local file named VERIFY exists (all the user needs to do is a "REWIND, VERIFY." before calling COMPILE).

Assembling Multiple Routines

If COMPILE is called with DECK=* specified, then the four UPDATE directives flagged with asterisks above (under "The UPDATE Process") are not included.

If the first character in "prog" is an asterisk (for example, PROG=*PFDECKS), then COMPILE does two things -- it removes the asterisk from the PROG value and sets DECK=*. This option is used when it is desired to modify and catalog multiple binarys on the same file. The PL parameter must be specified to name the SCOPE PL on which the programs reside. Example:

```
NV8,P5,T0.  ASSEMBLE PFM PP PROGS
99KK33,F+.
ATTACH,PROFIL,ID=NV8.
BEGIN,COMPILE,,*PFM,123,PL=PL18.

7-8-9

/CALL PFA
/CALL PFC
*COMPILE PFS
.                    (It is necessary to include /CALLs or
.                    *COMPILEs for each deck to be included;
.                    the value of PROG is used only as a name
                     for the binary file, and DECK=* is simulated.
```

Multiple COMPILE Executions

Multiple "BEGIN,COMPILE" executions may be stacked within one job. It is possible, for instance, to assemble the System Texts in one COMPILE step, and use them in assembling another program in a following step.

Special Programs

CMRS -- If PROG=CMRS is specified, all four CMRs are assembled (all on the file BCMRS). CMR1 and CMR3 are not listed. The CMR date must be specified as VER=mmddyy.

CMRn -- If PROG=CMRn (n=0,1,2,3), the corresponding CMR is assembled, listed, and put on the file BCMRn. If the CMR date is not specified, an assembly warning will occur, and the current date will be used.

CMR -- If PROG=CMR, the default CMR (currently CMR0) is assembled on the file BCMR. An assembly warning will occur, noting that the default CMR was used.

TEXTS -- If PROG=TEXTS is specified, the System Texts CPCTEXT through STATEXT are assembled on the file BTEXTS.

0SY -- If PROG=0SY is specified, the 844 Buffer Controlware is processed. The binary cards must be the input record, and the version must be specified via the VER parameter (e.g., VER=A10).

The /LOCAL Pseudo-UPDATE Directive

The UPDATE directives necessary for implementing a local modification resemble the following example:

```
/IDENT L999PROGR                                    A
/BEFORE DECKR.3                                     B
/IF -DEF,INSTALL                                    C
*IDENT L999PROGR                                    D
*INSERT FORPROGM.6                                  E
* L999PRCGR        ... <NAME> ... <DATE>
*
*          <COMMENTS>
...
...
<THE DIRECTIVES FOR THE MODIFICATION>
...
*COMPILE PROG                                       F
*/ END OF MOD L999PROGR                             G
/ENDIF                                              H
// END CF MOD L999PROGR                             I
```

The /LOCAL card causes the COMPILE procedure to provide the cards labeled. A through I, requiring the user to provide only the comments and directives for the modification. The format of the /LOCAL card can be any of the following:

```
/LOCAL L999
/LOCAL L999PRCGR
/LOCAL L999PRCGR,HDRPROGH
/LOCAL L999PRCGR,DECKR
/LOCAL L999PRCGR,HDRPROGH,DECKR
```

The local mod number (as L999) must be provided. The letter can be D, G, I, L, P, or Y. If the rest of the mod name (progr) is omitted, the value of the "prog" parameter (truncated to five characters) is used.

If the header mod name is omitted, the name is assumed to be HDR followed by the value of the "prog" parameter (truncated to six characters).

If the COMDECK name (deckr) is omitted, the value of the "deck" parameter is used.

If the user provides the *COMPILE card, the procedure won't; if the procedure provides the card, it will use the value of the "prog" parameter. The *COMPILE card must be the last card in the modification if the user provides it.

The first card after the /LOCAL card must be a comment card. COMPILE will insert the local mod name into columns 3-11 of this card. Columns 12-20 will be cleared. If columns 21-30 are empty, the value of the AC parameter will be inserted, and if columns 51-60 are empty, the current date will be inserted.

If the /LOCAL directive is used, all cards in the local mod will be punched. Multiple local mods, each prefaced by /LOCAL, can be present. Whenever /LOCAL is used, the listing from the LOCALPL UPDATE will be printed.

HEADER PROCEDURE

The HEADER procedure produces the UPDATE directives necessary when adding the first local modification to a program. A COMDECK and a "header mod" are produced.

A new COMDECK for the LOCALPL is created. This COMDECK will be used to contain all local modification code for the program, including code added in the future. The format of the COMDECK is

```
/COMDECK <COMDK>
*/ BEGIN MODS TO <PROG>
*/ END OF MODS TO <PROG>
```

The variable parameters "comdk" and "prog" are explained below. The COMDECK is never produced when we are introducing a modification to a SCOPE common deck; all local code for SCOPE common decks goes into one of the LOCALPL COMDECKS named SC4ACOM, SC4BCOM, IN4COM, or PL12COM, which already exist.

The "header mod" is a set of comment cards that is added near the beginning of every locally-modified program. It serves as a notice that the program is in fact locally modified. Every subsequent local modification will add its own set of comments after the header mod, so that a concise summary exists of all the local changes to the program. The format of the header mod cards is shown below:

```
/IDENT HDR<TPROG>
/BEFORE <COMDK>.3
*IDENT HDR<TPROG>
*INSERT <IDENT>.<SEQNO>
*****************************************
*                                       *
*        --LOCAL MODIFICATIONS--        *
*                                       *
*****************************************
*
*COMPILE <PROG>
*/ END OF MOD HDR<TPROG>
// END OF MOD HDR<TPROG>
```

The variable parameter "tprog" is "prog" truncated to six characters.

The HEADER procedure puts the COMDECK on the file DECKS, and the header mod is put on the file IDENTS. The COMPILE procedure processes these files, merging their contents into the set of UPDATE directives it builds. Two separate files are used in order to keep everything in sequence in case HEADER is called more than once; the files are not rewound, so that their contents are cumulative.

The COMDECK and the header mod are also written to the PUNCH file for subsequent punching. This can be disabled by including a "ROUTE,PUNCH,DC=SC." card as a final control card.

HEADER is executed as follows:

```
BEGIN,HEADER,,prog,comdk,ident,seqno.
```

The "prog" parameter is required; the others are optional. Each of the parameters are listed below:

prog -- the name of the program being modified. In cases where the binary deck name is different from the UPDATE PL deck name, the latter should be used.

comdk -- the name for the COMDECK. If omitted, "prog" is used as the COMDECK name. If the program being modified is a SCOPE common deck, one of the following must be specified as "comdk:" SC4ACOM, SC4BCOM, IN4COM, or PL1ZCOM.

ident, seqno -- these parameters describe the location in the program where the header mod is to be inserted. If "ident" is omitted, the value of "prog" is used; if "seqno" is omitted, "6" is used. The user must usually examine a listing of the program to be modified, in order to intelligently specify values for these parameters.

Examples

```
BEGIN,HEADER,,1AJ.
```

A header mod is inserted at 1AJ.6 and LOCALPL COMDECK "1AJ" is created.

```
BEGIN,HEADER,,1AJ,IDENT=SC41926,SEQNO=134.
```

A header mod is inserted at SC41926.134, which must be a card within 1AJ. The LOCALPL COMDECK "1AJ" is created.

```
BEGIN,HEADER,,INIT,SC4BCOM.
```

A header mod is inserted at INIT.6. Since INIT is a SCOPE common deck on PL1B, SC4BCOM must be specified as the COMDECK name. No new LOCALPL COMDECK is created.

DSBUILD PROCEDURE

The DSBUILD procedure builds a deadstart tape from the running system. If any permanent files containing transfer records are attached, the transfer records on those files will be used; otherwise, the transfer records will be taken from the running system. If necessary, additional EDITLIB directives are taken from input cards.

DSBUILD is executed as follows:

```
BEGIN,DSBUILD,,tape,option.
```

Both parameters are required. They are:

tape -- for production of deadstart tapes, the number of the tape (102, etc.); for test tapes, any 1-5 character name. The lfn and VSN of the created deadstart tape will be "A" followed by this parameter.

option -- must be SYSTEM to create a production tape, or TEST to create a test deadstart tape. When SYSTEM is specified, the DSCOMPARE procedure is automatically called and the ITEMIZE of the tape is cataloged on the EDITLB pack. Never use SYSTEM when building a test tape.

When invoked, the DSBUILD procedure pauses, waiting for an entry of n.YES; when this is entered, an EDITLIB,RESET is performed and then the operator is asked to run the daily EDITLIBs in order to ensure that the running system contains the correct EDITLIBs. The EDITLIB job should be killed or dropped when it displays "TURN SEC,EDITLIB,OFF."

By entering n.NO, the EDITLIB,RESET and EDITLIB insertions can be bypassed. This should not be done when building a production tape.

When building a test tape, you can do a compare against the current production tape by executing

```
BEGIN,DSCOMPARE,,tape.
```

The EDITLB pack is mounted by DSCOMPARE. DSCOMPARE is automatically called when building a production tape.

As an example, suppose it was necessary to build production Deadstart Tape 106, with changes to MTR, IRCP, and the CMRs. Something unusual is also being introduced: a new PP routine ALZ and a new CP routine SNXLP are being added. They cannot be EDITLIBed in because they will not run without a CMR change. They are both on the file BSNXLPALZ. The deck to build this tape would look something like this:

```
NXXYY,MT1,T0.
99ZZ99,FWD.
BEGIN,MOUNT,VSN=EDITLB.
ATTACH,BMTR, ...
ATTACH,BIRCP, ...
ATTACH,BCMRS, ...
ATTACH,BSNXLPALZ, ...
ATTACH,PROFIL,ID=NVB.
BEGIN,DSBUILD,,106,SYSTEM.
EXIT.
EXIT.
REQUEST,A106,RING,VSN=A106.
7-8-9
ADD(ALZ,BSNXLPA,AL=0)
LIBRARY(NUCLEUS,OLD)
ADD(SNXLP,BSNXLPA,AL=1,FL=30000,FLC=0)
FINISH.
6-7-8-9
```

Note that a dummy REQUEST card is necessary to satisfy tape staging.

Both the DSBUILD and the DSCOMPARE procedures have a verify mode, as described in the COMPILE procedure user's guide.

The SCOPE transfer records are listed below:

CEA - CES	Deadstart Initialization
PC1 - xxx	C.E. Diagnostics
CED - DTS	CONTROL
CMR0 - CMR3	CMRs
COM - P	CONTROL
OSY	844 Buffer Controlware
STL	PP Resident
IRCP	Deadstart CP Routine
MTR	Monitor
DSD	System Display (only the main overlay is a transfer record; DSBUILD correctly puts the remaining overlays into PPLIB)

APPENDIX B

SYSTEM UTILITIES USED

IN INSTALLING A LOCAL MODIFICATION

APPENDIX B

SYSTEM UTILITIES USED IN INSTALLING A LOCAL MODIFICATION

A local mod is written as a source language correction set that, through several steps, becomes implemented as a change to the running operating system. The SCOPE utility programs used in this process are discussed on the following pages; the utilities are

- UPDATE
- Language Translators
- LOADER
- EDITLIB

UPDATE - SOURCE LIBRARY MAINTENANCE UTILITY

The UPDATE program is used to maintain files containing source programs. These Program Library files, or PLs, are implemented in a random-access compressed card image format which allows quick access and efficient storage utilization. An UPDATE PL contains three types of information: DECKs, COMDECKs, and IDENTs.

A DECK is a source program module. Each DECK has a name, and each card within a DECK has a unique number, which is invariant. A correction set may delete cards from a DECK, but these cards remain on the PL so that it can always be restored to a previous state.

A COMDECK is a group of source statements which may be repeated in several program modules. UPDATE allows these statements to be entered once onto the PL as a COMDECK, which is referenced by each DECK that requires the common code. Each COMDECK has a unique name, and the cards are assigned numbers.

IDENTs are sets of modifications to DECKs or COMDECKs. An IDENT may delete cards from or add cards to a DECK, COMDECK, or another IDENT. Cards that are added are identified by the unique IDENT name and a sequence number within the IDENT. IDENTs are also called "correction sets."

An UPDATE run (Figure B-1) is controlled by control card parameters on the UPDATE command and by input directives. Usually, only DECKs named on an input directive (and COMDECKs called by these DECKs) are output to the source language file (the "COMPILE" file). The COMPILE file is usually used as input to a language translator.

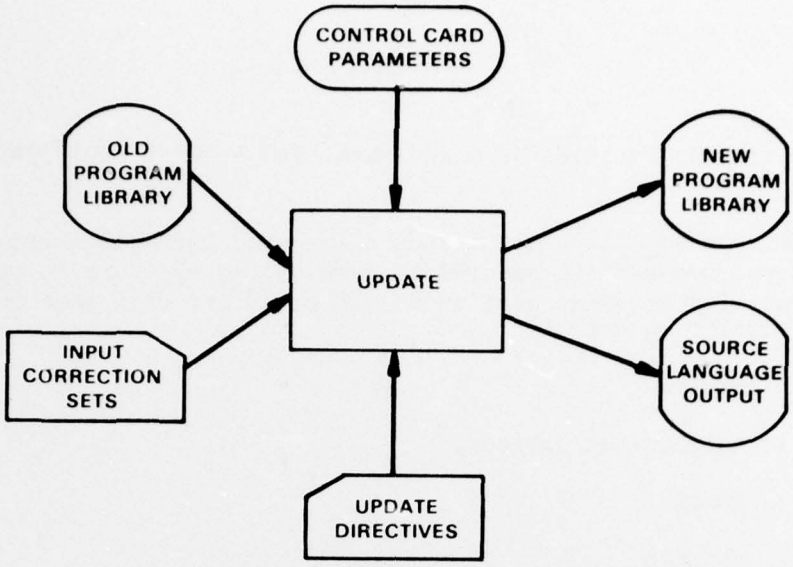


Figure B-1. The UPDATE Process

LANGUAGE TRANSLATORS - FTN AND COMPASS

The SCOPE 3.4 operating system is written in CDC Extended FORTRAN (FTN) and in assembly language (COMPASS). The FTN compiler accepts intermixed COMPASS subprograms. The translation process (Figure B-2) converts a group of source subprograms (from cards or from an UPDATE COMPILE file) into a file containing the relocatable binary object code of the subprograms. The object file is processed further by the LOADER. Translation is controlled by control card parameters on the FTN or COMPASS command. Auxiliary source input to COMPASS programs from systems text overlays is often required.

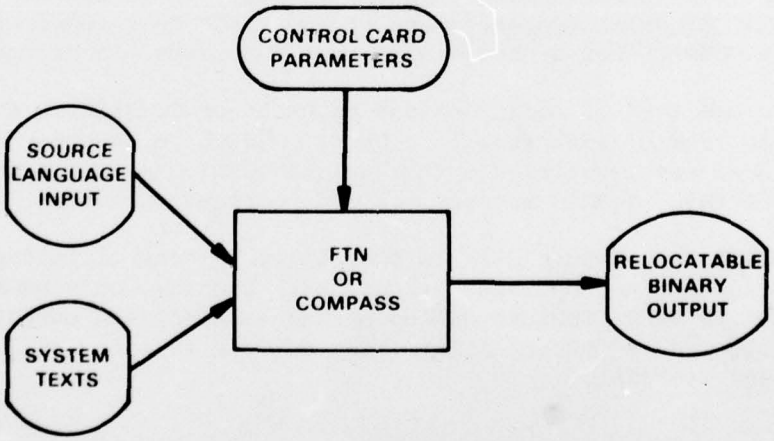


Figure B-2. The Translation Process

THE LOADER UTILITY

The local mod installation process uses the CYBER LOADER to convert the relocatable binary object programs, produced by the language translators, into linkage-edited absolute overlays. External references are resolved by including subprograms from object librarys where necessary. Multiple-overlay program building is controlled by input directives; the entire absolutization process (Figure B-3) is controlled by control card parameters on LIBRARY, MAP, LDSET, LOAD, and NOGO commands. The absolute overlay file is used as input to the EDITLIB program.

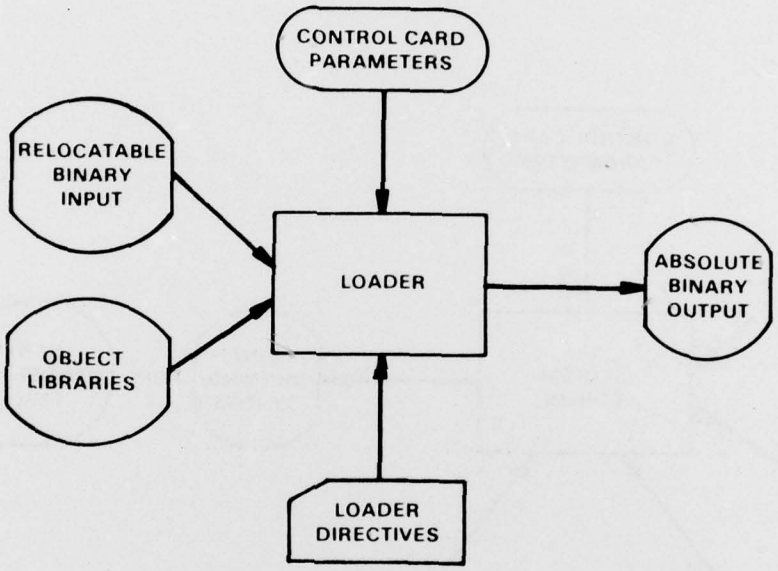


Figure B-3. Conversion of Relocatable Programs to Absolute Overlays

EDITLIB - OPERATING SYSTEM MODIFICATION UTILITY

The System EDITLIB program integrates changes into the SCOPE 3.4 operating system. Two modes of operation are featured -- modification of the running system and creation of a modified deadstart tape. Changes to the running system are temporary, lasting only until the first subsequent deadstart.

Creation of a deadstart tape containing a local modification makes the mod a permanent part of the operating system, and is one of the final steps in the installation of a local mod.

Execution of the EDITLIB program (Figure B-4) is controlled by control card parameters on the EDITLIB command, by input directives, and by console interaction with the computer operator.

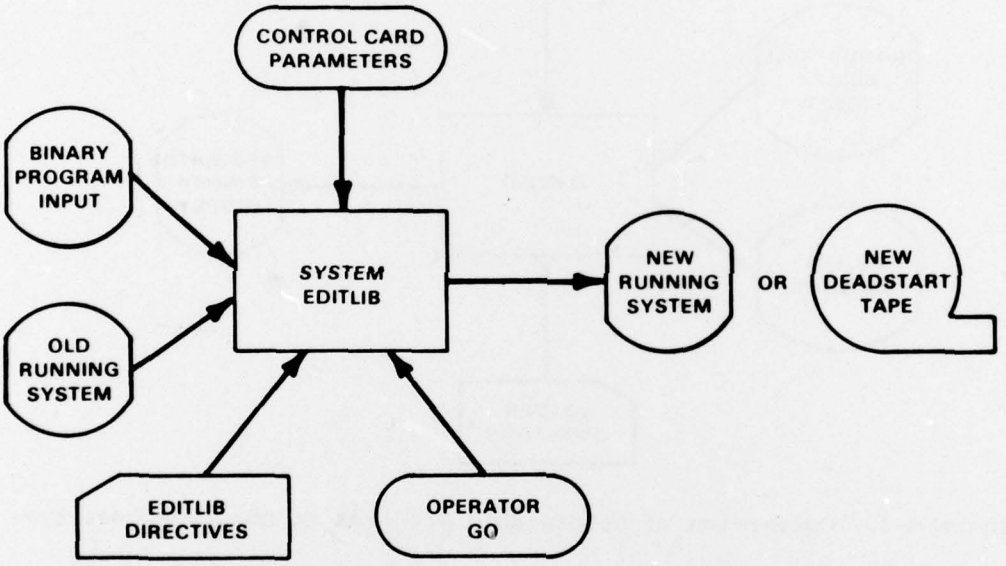


Figure B-4. EDITLIBing the Program into the System

DISTRIBUTION

Commanding Officer
NAVAL COASTAL SYSTEMS CENTER
Panama City, FL 32407
ATTN: J.D. Brown (Code 732)

Director
NAVAL RESEARCH LABORATORY
Washington, DC 20375
ATTN: Code 1721

Commanding Officer
NAVAL AIR DEVELOPMENT CENTER
Warminster, PA 18976
ATTN: Code 50
Code 85

Commanding Officer
FLEET NUMERICAL WEATHER CENTRAL
Monterey, CA 93955
ATTN: Code 006

Officer in Charge
NAVAL UNDERWATER SYSTEMS CENTER
New London, CT 06320
ATTN: Richard Whittaker (Code 4421)

Commander
NAVAL OCEAN SYSTEMS CENTER
San Diego, CA 92152
ATTN: Ken Medin (Code 9121)

Commander
DAVID W. TAYLOR NAVAL SHIP RESEARCH
AND DEVELOPMENT CENTER
Bethesda, MD 20084
ATTN: Lorraine Minor (Code 1892.3)

Commander
NAVAL WEAPONS CENTER
China Lake, CA 93555
ATTN: Code 5132

Commanding General
AIR FORCE WEAPONS LABORATORY (ADP)
Kirtland AFB
Albuquerque, NM 87117
ATTN: Software Section

DISTRIBUTION (Continued)

Commanding General
EGLIN AIR FORCE BASE, FL 32542
ATTN: Mr. Eddie Blackwell (Code ADTC/ADDSS)

University of Arizona
UNIVERSITY COMPUTER CENTER
Tucson, AZ 85721
ATTN: Steve Jay

FLUOR CORPORATION
3333 Michelson Drive
Irvine, CA 92730
ATTN: Mr. Thomas N. Burt

Mr. Frank Vince
CONTROL DATA CORPORATION
P. O. Box 0-HQS10D
Minneapolis, MN 55440

Burnie Meyer
CONTROL DATA CORPORATION
6003 Executive Blvd.
Rockville, MD 20852

Thomas L. Hank
CONTROL DATA CORPORATION
4201 Lexington Ave. N.
Arden Hills, MN 55112

SYSTEMS AND DEVELOPMENT GROUP
CONTROL DATA CORPORATION
4201 Lexington Ave. N.
Arden Hills, MN 55112

Edward O. Minasian
2051 28th Avenue
San Francisco, CA 94116

Mr. Art Hartley
CONTROL DATA CORPORATION
4201 Lexington Ave. N.
Arden Hills, MN 55112

Mr. John L. Wardell
CONTROL DATA CORPORATION
4201 Lexington Ave. N.
Arden Hills, MN 55112

Mr. James Whitlock
Office of Computer Services
STATE UNIVERSITY OF NEW YORK
AT BUFFALO
4250 Ridge Lea Road
Buffalo, NY 14226

DEFENSE DOCUMENTATION CENTER
Cameron Station
Alexandria, VA 22314 (12)

LIBRARY OF CONGRESS
Washington, DC 20540
ATTN: Gift and Exchange Division (4)

Local:

E41
F10
K
K50
K60
K70
K74
K74 (Zirkle) (40)
N20
V
X210 (2)
X2101 (GIDEP) (2)